PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 :
G06F 9/26, 12/06

A1

(11) International Publication Number: WO 94/12929

(43) International Publication Date: 9 June 1994 (09.06.94)

(21) International Application Number: PCT/US93/10836

(22) International Filing Date: 10 November 1993 (10.11.93)

(30) Priority Data:
07/980,060 23 November 1992 (23.11.92) US

(71) Applicant: S-MOS SYSTEMS, INC. [US/US]; 2460 North First Street, San Jose, CA 95131 (US).

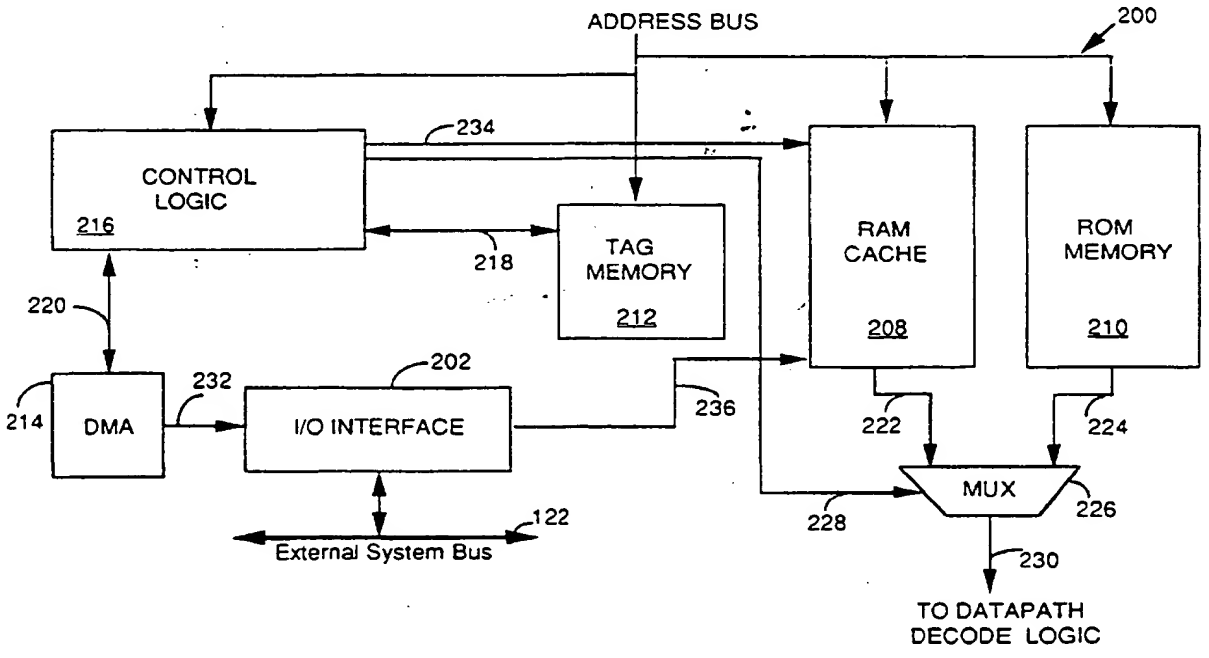
(72) Inventors: DEMERS, Eric; 900 High School Way, Apt. 2228, Mountain View, CA 94041 (US). LENTZ, Derek; 17400 Phillips Avenue, Los Gatos, CA 95032 (US).

(74) Agent: WERNER, Raymond, J.; S-MOS Systems, Inc., 2460 North First Street, San Jose, CA 95131 (US).

(81) Designated States: JP, KR, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: A MICROCODE CACHE SYSTEM AND METHOD



(57) **Abstract**

A microcode caching system for storing microcode for use by a digital system that processes program instructions to perform functions. The system comprises a read only memory (ROM) configured to store a first group of the microcode, and a random access memory (RAM) cache configured to temporarily store blocks of a second group of the microcode. The second group of the microcode is directly mapped into the RAM cache from a memory device separate from the RAM cache so that the blocks can be swapped into and out of the digital system. The microcode caching system is integrated with the digital system, whereas the separate memory device is not.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

A MICROCODE CACHE SYSTEM AND METHOD

BACKGROUND OF THE INVENTION

1. *Field of the Invention*

The present invention relates to digital processors, and more particularly, the present invention relates to a random access memory (RAM) cache for storing microcode instructions.

2. *Related Art*

Microcode, or microprogramming, was first proposed as a way of making the logic which is used to control the function of a central processing unit (CPU) more regular. It enables a simple datapath to perform a complex program instruction, and it achieves this by breaking down the functions necessary to execute the complex instruction into a series of simple operations which are performed sequentially. Microcode is not to be confused with program instructions that correspond to the instruction set of the CPU.

A microcode address is formed from some or all of the contents of an instruction register, together with some state values which are internal to a microcode engine (or microsequencer). The microsequencer executes the microcode in order to control the data path. The advantage of microcode is that the control logic is implemented in a very regular structure, usually a read only memory (ROM). Changes or additions to the instruction set require that bits in the ROM be altered, or more ROM added. Others have proposed that for even greater flexibility, all or part of the ROM could be replaced by writable RAM. The RAM approach, however, is still limited in the amount of microcode control that can be achieved, because RAM is expensive to integrate on-chip, and has a low bit storage-to-chip area ratio compared to on- chip ROM. (Currently, ROMs have storage-to-chip area ratios that are 3:1 or 4:1 compared to RAMs, i.e., RAMs are 3-4 times larger than ROMs for the same amount of storage capacity.)

Some microcoded machines allow micro programs to use branches and subroutines, and the microsequencing logic can become very complex. A few techniques used in microprogramming have been adopted by designers of RISC processors, but this time at the user program level. These borrowed techniques include delayed branches and software managed pipeline interlocks.

Microcode exposes the programmer to pipeline and hardware details, which can be both advantageous and disadvantageous. Using microcode is more efficient and at the same time, harder to program. One major advantage is that multiple operations can be performed in one cycle if efficiently microcoded.

Because ROMs have very good access time, frequently used ROM operations will run fast if they are microcoded. An early RISC exponent pointed out that instruction cache memories also have good access times. Whereas ROM based microcode only contains a static set of operations chosen by the original designer, a conventional instruction cache can contain a dynamic set of frequently used operations selected automatically by the hardware to suit the current task. Conventional instruction caches, however, have not been used to store microcode. (Numerous cache memory configurations that have been used in various commercial computer systems are discussed by Stephen B. Furber, *VLSI RISC Architecture and Organization*, (Marcel Dekker, Inc., 1989); and John L. Hennessy *et al.*, *Computer Architecture - A Quantitative Approach*, (Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990).

The RISC movement has been seen as a reaction against microcode. This may be misleading. It is the complex instruction sets which are called into question, not the microcode implementation. Although it is true that in small machines microcode is needed to support complex instruction sets, it is equally possible to microcode a simple instruction set. Early RISC designs avoided the use of microcode, but some later commercial RISC designs have reintroduced it. The RISC trend to single cycle execution of most instructions will preclude the use of the more exotic microcode constructs. With a single cycle instruction, the microcode ROM is really just a regular decode structure.

In his text, Furber explains that digital processors with complex instruction sets typically use microcode to allow the complex instructions to be executed on a relatively simple datapath in several sequential steps, such as DEC's VAX-11/780. Here the advantage gained by the microcode over a subroutine of simple instructions is that the microcode is held in a memory with a very short access time compared with main memory. If the CPU has a cache memory for the program instructions, however, this advantage is lost. Microcode gives good performance to a pre-selected set of sequences of simple instructions, whereas an instruction cache will give the same performance to a dynamically self-adjusting set of sequences.

A prior VAX design used off-chip RAM to store microcode. The microcode was directly mapped into the chip. However, additional pins were required to interface with the off-chip RAM to access the microcode. Another drawback was the need for

the programmer to write code to fetch the necessary microcode.

What is desired is still greater flexibility in a CISC or RISC system that takes advantage of the speed of both microcode and caching techniques.

SUMMARY OF THE INVENTION

The present invention is directed to a caching system and method comprising both an on-chip ROM and a RAM cache for storing microcode.

The present invention is flexible enough to look for microcode in either the ROM or the RAM cache using tag information and information based on the target address decoded at the time the corresponding instruction is decoded. The present invention permits critical microcode to be stored in the ROM, and additional microcode to be cached into and out of the RAM on an as needed basis. The present invention further permits off-chip diagnostic microcode to be cached into the microcode RAM cache to test the chip or a peripheral device, for example.

The present invention uses the chip's standard I/O channels to access a group of microcode limited in size only by the amount of available off-chip memory or the microcode address space. From the programmer's point of view the microcode RAM cache of the present invention is transparent. The programmer does not have to write code to fetch the necessary microcode. The present invention is thus faster and more efficient than the prior designs. A direct mapped cache can be used because the microcode can arrange code to avoid cache thrashing. Other cache mapping techniques are also envisioned.

The present invention also conserves clock cycles. Because of current technology, clocks speeds have decreased access times such that off-chip RAM storage of microcode is no longer useful. A 50 MHz clock translates into a 20 ns clock cycle, therefore the time to get off and back on chip with the microcode eats up too many clock cycles to be efficient. A solution would be to use very fast off-chip RAMs, but such devices are cost prohibitive and require many dedicated pins on the package. The present invention can use slower, less expensive SRAMS or DRAMS for the on-chip cache and still save valuable time in the form of clock cycles.

Another advantage of the present invention is that new microcode can be added to the system, and therefore can run at the speed of the built-in ROM microcode, with only the overhead of loading new microcode when needed.

A still further advantage is that bugs can be fixed using RAM microcode or a combination of ROM and RAM microcode, such as ROM subroutines that branch to RAM, for example. Other patching techniques between RAM and ROM and vice-

versa are also envisioned.

The foregoing and other features and advantages of the present invention will be apparent from the following more particular description of the preferred embodiments of the invention, as illustrated in the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood if reference is made to the accompanying drawings in which:

FIG. 1 is a block diagram of a conventional microcode storage system.

FIG. 2 is a high level block diagram of a microcode caching system of the present invention.

FIG. 3 is representative diagram showing microcode stored in blocks of main memory.

In the drawings, like reference numbers indicate identical or functionally similar elements. Additionally, the left-most digit of the reference number identifies the drawing in which the reference number first appears.

DETAILED DESCRIPTION

Introduction

The present invention was developed for a single (semiconductor) chip graphics processor having a relatively small instruction set, but very complex microcode. The problem that arose was how to speed-up processing of program instructions in the data path. Solving this problem was critical to the success of the graphics processor because the number of functions required to be performed to execute each graphics program instruction is formidable.

Hundreds to perhaps millions of floating point calculations are required to execute one graphics program instruction. In a graphics processor, for example, a large burden is therefore placed on the microcode to perform the necessary functions to execute the program instruction. In contrast, execution of a typical CISC or RISC instruction may require only the addition of two numbers.

DEC's VAX-11/780, which was introduced in 1978, had a pre-RISC architecture using complex instructions and microcode for data path control. The microcode was partly fixed and partly writable. The writable section was used to implement some instructions and to patch others, and to enable instructions to be built for diagnostic purposes. A simplified block diagram of the VAX-11/780 is shown in FIG. 1. A CPU 102 includes a processor 104, fixed microcode block 106 and writable block 108. Also shown in FIG. 1 are the following: instruction path 110; data

read and write paths 112 and 114, respectively; data cache 116; write buffer 118; virtual-to-physical address converter, or translation look aside buffer (TLB) 120; external system bus 122; memory subsystem (main memory) 124; and I/O subsystem 126. The operation of the elements are conventional. It is noted, however, that data cache 116 is only used for caching data into processor 104.

System block diagram

The present invention will now be discussed in connection with a graphics processor chip, however, the present invention can be applied to any system using microcode, as will become evident to those skilled in the art. For example, a digital signal processor performing Fast Fourier Transformations can utilize microcode to perform the many required add and multiply operations. The present invention is therefore not limited to a graphic processor per se.

The terms processor, CPU, and digital processor are often used interchangeably in this field. The term "processor" is used hereafter with the understanding that other similar terms could be substituted therefore without changing the underlying meaning of this disclosure.

The terms chip, integrated circuit, semiconductor device and microelectronic device are often used interchangeably in this field. The present invention is applicable to all of the above as they are generally understood in the field.

A preferred embodiment of an integrated microcode storage system of the present invention is shown at 200 in FIG. 2. Integrated microcode storage system 200 includes a microsequencer 201, also called a microcode engine, that executes microcode instructions in datapath decode logic in order to control a data path (datapath decode logic and data path not shown).

Microsequencer 201 executes a predetermined microcode routine depending on the function specified by the program instruction. Microcode is executed line-by-line. Each line of microcode is decoded to control the functional blocks (registers, multiplexers and the like) that comprise the data path.

Referring again to FIG. 2, a standard input/output (I/O) interface 202 is shown connected to an external system bus 122. An internal address bus is shown at 206. A random access memory (RAM) block 208, a read only memory block 210, a cache tag memory block 212, a control logic block 216 and a direct memory access (DMA) controller block 214 comprise the major elements of system 200. System 200 also interfaceses with a main memory that is separate from system 200 (e.g., off-chip).

DMA 214 is used for fetching microcode from a main memory, or the like (not

shown), by translating micro instruction addresses into memory addresses. Alternatively, the functionality of DMA 214 could be built into microsequencer 201.

Control logic block 216 includes cache miss logic to handle cache misses that may occur when RAM cache 208 is accessed. The cache miss logic, together with cache tag memory block 212, perform standard caching techniques to update the microcode stored in RAM cache 208. Many conventional caching techniques, such as those discussed in Hennessy *et al.*, can be used in system 200, as will become evident to those skilled in the art. The microsequencer also includes a next microcode address generator (not shown) that selects which memory to use (either RAM 208 or ROM 210) based on the next address (i.e., the next address will be in the RAM's address range or in the ROM's address range). The next address generation is based on the current address, data from the datapath, and/or the datapath decode logic (typically it will be the output of RAM 208 or the ROM 210).

Memory 208 or 210 then outputs the microcode data for that cycle via buses 222 and 224, respectively, to a multiplexer (MUX) 226. Control logic block 216 generates a MUX select signal and sends it to MUX 226 via line 228 to output the microcode data to the datapath decode logic via bus 230.

This microcode data is decoded to control the datapath and to tell next address generator which address source to use for the next address (either the address comes from the datapath, the microcode data out of the ROM/RAM or it is just the next sequential location).

If the next address is a valid RAM address and RAM 208 contains the correct information at that address, then there is a cache hit. Otherwise, the cache miss logic in control logic block 216 freezes the datapath and the microsequencer.

The cache miss logic then asks DMA 214 (via a bidirectional bus 220) for the missing information. DMA 214 fetches this information and stores it in RAM 208. To do this, DMA 214 communicates with I/O interface 202 via unidirectional bus 232. I/O interface 202 accesses the main memory via external system bus 122. The requested microcode data is sent back via external system bus 122 to I/O interface 202. The cache miss logic, that initiated the request, tells RAM cache 208 to expect the microcode data via a unidirectional line 234. I/O interface 202 sends the new microcode data to RAM cache 208 via a bus 236. Tag memory 212 is then updated with the new information and the system continues where it was prior to the cache miss.

Alternatively, when a cache miss occurs, the sequencing device merely sends a microcode block address with the request. The microcode block address can he

translated into any other form to fetch it from main system memory. In a virtual address scheme (such as is commonly used in processors) or any other sort of scheme can be used with the microcode. for example, DMA 214 receives the block address, it translates it to a word address and adds a pointer to it; consequently the main memory of the system can be organized into 32-bit words and the microcode can be located starting at any word aligned location within that space. The only requirement that the amount of space required to store the RAM code should be continuous. It is preferred that the cache blocks be aligned to make use of burst mode access on any interconnect buses between the microcode cache and external microcode memory.

In a system employing virtual memory, it may be useful to allow the RAM microcode to exist in virtual memory space; this implementation will become evident to those skilled in the art. A standard TLB circuit is required which translates the physical RAM microcode address to a virtual address in the system. The hardware could be built to understand the system page table formats. Alternatively, the host processor could be interrupted to service the request. It is also possible that a system could use a special format for the microcode page table and allow the hardware to have its own (possibly simpler) page table format.

The size of RAM cache 208 and ROM 210 are implementation specific, but for the purposes of discussion the RAM comprises 32 blocks of 8 microwords. Microcode RAM cache 208 is implemented in a direct mapped format. This permits large amounts of microcode to be stored in main memory. A realistic amount of off-chip addressable microcode is 128 Kbytes.

The address locations in microcode RAM cache 208 are directly mapped to a section of main memory (off-chip memory, or the like) that is substantially larger than the RAM cache itself. Blocks of the microcode stored in main memory can therefore be swapped into and out of the microcode RAM cache when a cache miss occurs. Other conventional cache mapping schemes can be adapted for use with the present invention, as will become evident to those skilled in the art.

Control logic 216 is configured to detect addresses corresponding to microcode requests that are directed toward RAM cache 208 on address bus 206. Control logic 216 receives information from cache tag memory block 212 along a bidirectional bus 218, or the like, as to the validity of the RAM location associated with the desired address detected on address bus 206.

If RAM cache 208 contains an invalid address, control logic 216 stops all the data path elements under microcode control and requests, via I/O interface 202, the missing microcode from DMA 214. Control logic 216 then passes the block address

associated with the desired microcode's address to DMA 214 via a bidirectional bus 220, or the like.

DMA 214 is configured to make requests to the system's main memory for the needed microcode. Microcode is stored in main memory in blocks 302 as shown generally in FIG. 3. DMA 214 stores a beginning main memory location 304-310 corresponding to each main memory block containing microcode. DMA 214 uses a pointer to offset the base block address to locate the requested microcode in the block of microcode returned by the main memory. Because DMA 214 uses a base pointer for the address of the block requested, microcode can be stored at virtually any location in system memory. All that is required is that the base pointer be changed to reflect the base address of the microcode. Alternatively, the microsequencer can be modified to generate the main memory address, rendering a DMA unnecessary.

When an address is passed to DMA 214 by control logic 216, DMA 214 adds a base pointer address value to the block address, and fetches the block from the main memory in a known fashion via I/O interface 202.

The datapath and decoding is frozen until the requested microcode is received into the cache. In a preferred embodiment, the datapath is fully pipelined so the best way to stop execution is to gate or "freeze" the main clock buffer for gated logic (not shown). Freezing requires that the clocks to all the circuits except those used to load the microcode RAM be paused. The sequencing elements of the device then requests the missing code from external memory and restarts the clocks when the code is retrieved, as described above. I/O interface 202 subsequently receives the microcode information from main memory and passes it on to RAM cache 208 via data bus 207.

In a preferred embodiment of the present invention, critical or more frequently used microcode is stored in ROM 210 and less critical or infrequently used microcode is cached in the on-chip RAM 208. Good system performance is achieved by optimizing where specific microcode is stored. Requests for microcode in ROM 210 are verified by control logic 216. The microcode is subsequently read out of ROM 210 and placed on data bus 207 for use by the microsequencer to control the data path.

Fabrication of the chip is also sped up, because microcode that takes longer to write and debug does not have to be stored in the ROM 210. Later refined microcode is simply stored in main memory and cached to the microcode RAM cache 208 when needed.

In addition, infrequently used special test or diagnostic microcode does not have to be stored in ROM 210 and take up valuable space. Special test or diagnostic

microcode can be cached in on an as needed basis. Furthermore, the special test or diagnostic microcode can be written at any time and is not limited by the size of ROM 210.

The program instruction format is also flexible because of the microcode RAM cache 208 of the present invention. Program instructions can be easily modified (e.g., retargeted) to have the system look to ROM 210 or RAM cache 208 for the corresponding microcode. If the original ROM-stored microcode is problematic or obsolete, new microcode can be added and the program instruction changed to direct the system to look to RAM cache 208 rather than ROM 210 for the necessary microcode.

Branching from ROM to RAM and vice-versa

In a preferred embodiment of the present invention, the ROM and RAM are mapped to different locations in a continuous main memory space so that they can be used in an identical manner. The only difference between the two is that they are mapped to different addresses. If the whole memory space is represented by a n bit address, some of the address space can be mapped to RAM, while other address space can be mapped to ROM. For example, given a 14 bit address space, the ROM is allocated a 13-bit address space, and so is the RAM. Consequently, the most significant bit of the address is the ROM/RAM select (i.e., when the bit is set, the RAM is selected).

The current implementation for branching to and from ROM/RAM requires that pre-decode logic be added to the system to assure that most of the time only one memory is dissipating power. The pre-decode logic is designed to monitor the current usage of ROM/RAM (i.e., it monitors the ROM/RAM select bit). It also must monitor "possible" future addresses (e.g., branch addresses). When the pre-decode logic detects that a switch can occur between the ROM and RAM, it enables the other memory unit and subsequently disables the latter. For example, if the RAM is currently being used, it is enabled; if a microcode line then executes a branch to ROM, the pre-decode logic enables the ROM before the line is actually executed. The above is performed by doing a branch-command look-ahead in the microcode.

Of course, to turn off a unit, the pre-decode logic must determine that the unit will not be in use in the "near" future and the unit must not currently be in use.

The programmer can use subroutines to save program space and development time. Consequently, branching between the RAM and ROM is trivial so that a subroutine can be located in RAM as well as in ROM. The most common case is that a RAM routine will execute a ROM subroutine to save time and space. The space

saving comes from the fact that the subroutine in question does not have to be loaded into the RAM. The time saving comes from the fact that no cache miss has to occur in the RAM to execute the code; the ROM can always be used without any loss of time due to fetching non-resident microcode.

Special format used

When an instruction is sent to the processor, the address of the microcode to execute this instruction is include with it. This allows for patches to the instruction (which would then be in RAM) end simpler decode logic. Patches are made by changing the starting address of the microcode in the instruction. The processor software uses a table associated with the microcode when issuing instructions.

When microcode is modified a new instruction look-up table is created. The instruction format contain two fields relevant to this discussion. An "instruction number" field is not strictly required, but is useful for adding fixed hardware encoded instructions such as register and DMA controller instructions. A "microcode address" field holds the starting address of the instruction in the microcode address space. The software that drives the hardware reads and uses this table when booting or restarting.

The instruction number field can be associated with any starting address and is generally used to specify the instruction to be executed. The table is used to supply the starting address for each instruction. This technique is similar to dynamic linking in software.

Multiple sets of microcode can actually be implemented. For example, testing microcode can be used for boot diagnostics and then replaced by functional microcode sets.

Special testing code

Special code (e.g., for testing) can be easily executed, because the address of the instruction is given with the instruction, and the desired testing code is fetched from outside the processor. This is done by having the processor jump to a predetermined RAM address, and then feeding the processor the test code as it is required. The code can be as long as necessary (e.g., it could be as long as 13 bits of addressing, or longer with special fetching).

According to the present invention the processor can have a special testing mode. Externally accessible registers can be added to control a test mode. When this test mode is enabled, the normal execution of the processor is stopped and the processor enters a test mode.

Three registers are made available in one embodiment of this test mode. The first register is an address or an address/control register. Because the ROM and RAM are mapped to the same address space, any address can be loaded in the address register to access the required ROM or RAM word. The functions of six example fields for the address/control register are listed below in Table 1. Other bits could be used for additional testing functions as will become evident to those skilled in the art. The second register contains the data that the user wishes to write into the RAM on his cycle (the RAM address is contained in the address register). The third register contains the data at the ROM, RAM or TAG memory current address; if the RAM was previously written, this register should contain the same data as previously written, after an adequate delay. When test mode is disabled, the address loaded into the address register is the address at which the sequencer re-starts the microcode.

Microword Select Field	Selects which part of the wide microcode word is read or written to.
R/W Field	Indicates that a read or write is to be executed.
RAM/ROM Select Field	Selects the RAM or the ROM.
Tag Select Field	Selects the TAG RAM, RAM or ROM as determined by the RAM/ROM select field.
Test Enable Field	Enables or disables testing mode.
Test Micro Address Field	Selects the address to access in microcode RAM, ROM or tag. This field holds the starting address of microcode when testing is disabled.

Table 1.

While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example, and not limitation. Thus the breadth and scope of the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

CLAIMS

What is claimed is:

1. A caching system for storing microcode for use by a digital system that processes program instructions to control the digital system, wherein each program instruction comprises a plurality of instruction fields, the system comprising:

a read only memory (ROM) configured to store a first group of the microcode; and

a random access memory (RAM) cache configured to temporarily store subsections of a second group of the microcode, said second group of the microcode being mapped into said RAM cache from a memory device separate from said RAM cache, wherein said RAM cache is configured to swap said subsections into and out of the microcode cache system.

2. The system according to claim 1, further comprising means for receiving a request for microcode and determining whether the requested microcode is located in said ROM, said RAM cache, or the separate memory device.

3. The system according to claim 1, wherein the microcode caching system is on a single chip and the separate memory device is not on said chip.

4. The system according to claim 2, wherein the digital system has a microcode address range; and

said means for determining whether the microcode is located in said ROM or said RAM uses a first subsection of said microcode address range for said ROM and a second subsection for said RAM.

5. The system according to claim 2, wherein said means further comprises:

control logic for determining whether requested microcode is located in said ROM or in said RAM cache; and

a tag memory for indicating whether the desired microcode requested is actually resident and valid in said RAM cache when requested.

6. The system according to claim 5, wherein the system further comprises:

a direct access memory controller for fetching the desired microcode from the separate memory device and storing it in said RAM cache.

7. The system according to claim 1, wherein said second group of the microcode is direct mapped into said RAM cache from the separate memory device.

8. The system according to claim 1, wherein said second group of the microcode is substantially larger than said first group of the microcode.

9. The system according to claim 1, wherein the digital system is a graphics processor.

10. The system according to claim 1, wherein a subset of said first and second groups of microcode are used to perform diagnostics.

11. A method for storing microcode for use by a digital system in a ROM memory unit and a RAM memory unit, the ROM and RAM have m and k addressable locations, respectively, and are integrated with the digital system, wherein the digital system has an n-bit address space, comprising the steps of:

mapping all but one bit of addressable ROM locations to the address space; and

mapping all but one bit of addressable RAM locations to the address space;

wherein the remaining bit of the m bits and k bits is used as a select bit to access either the ROM or the RAM.

12. The method according to claim 11, further comprising the step of checking the value of the select bit to thereby monitor the current usage of the ROM and the RAM.

13. The method according to claim 11, further comprising the steps of:

predecoding branch addresses of the microcode to detect future switches between the ROM and the RAM; and

enabling the memory unit not being used and subsequently disabling the other if a switch is detected.

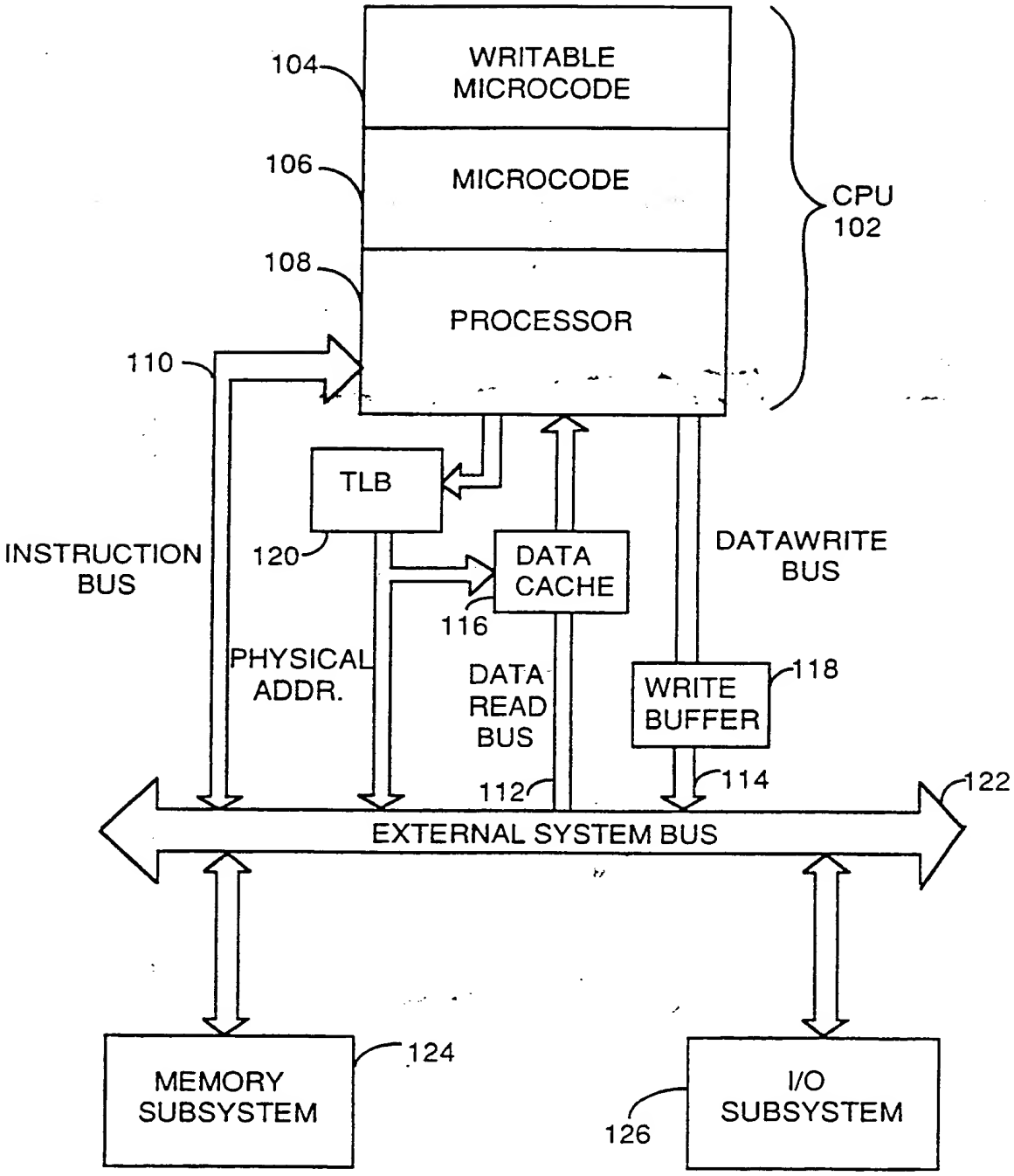


FIG. 1
(PRIOR ART)

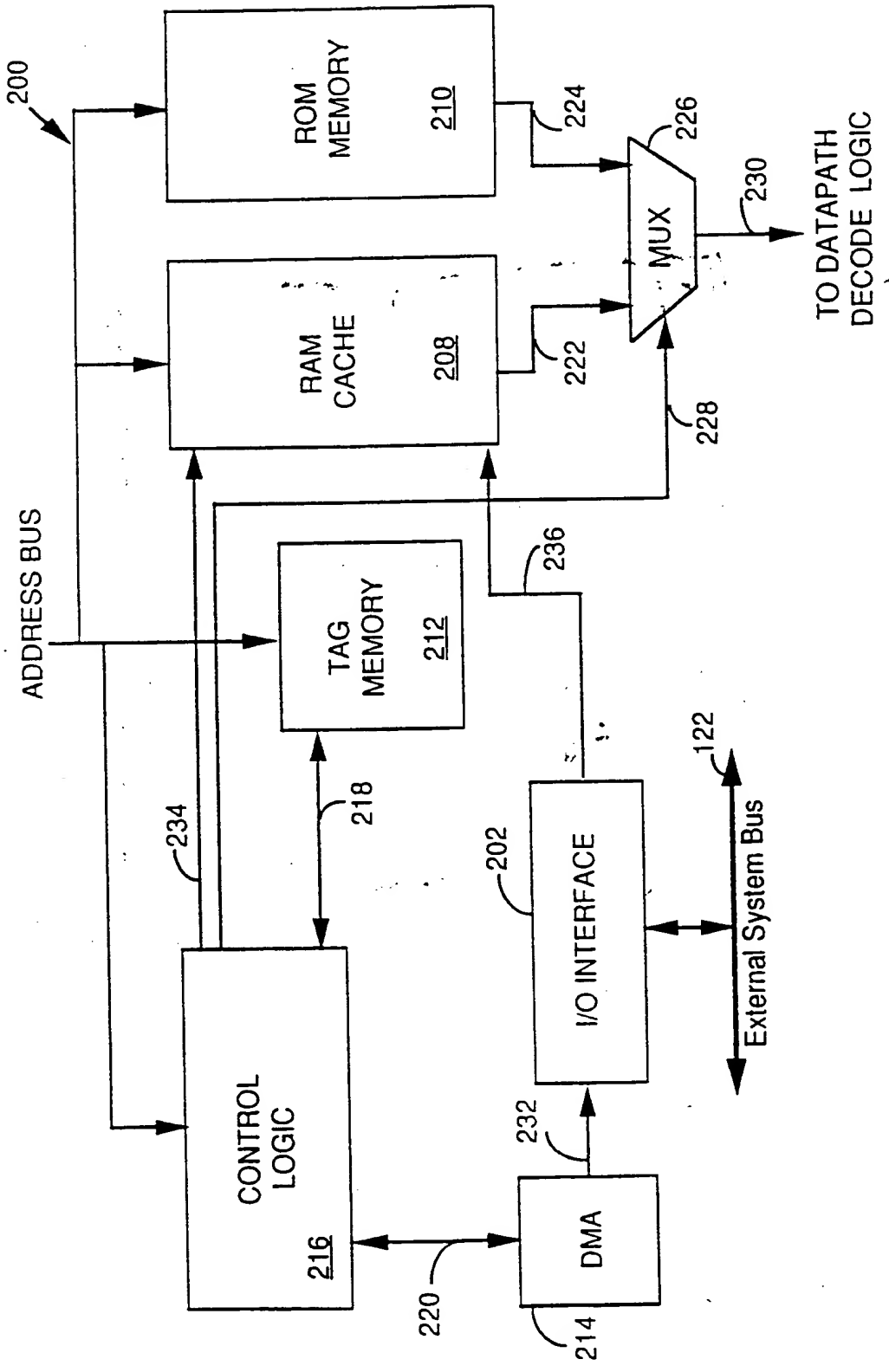

200
ROM MEMORY 210
224
226
MUX
230
TO DATAPATH DECODE LOGIC
RAM CACHE 208
222
228
ADDRESS BUS
TAG MEMORY 212
236
234
218
202
I/O INTERFACE
122
External System Bus
CONTROL LOGIC 216
232
DMA
220
214

FIG. 2

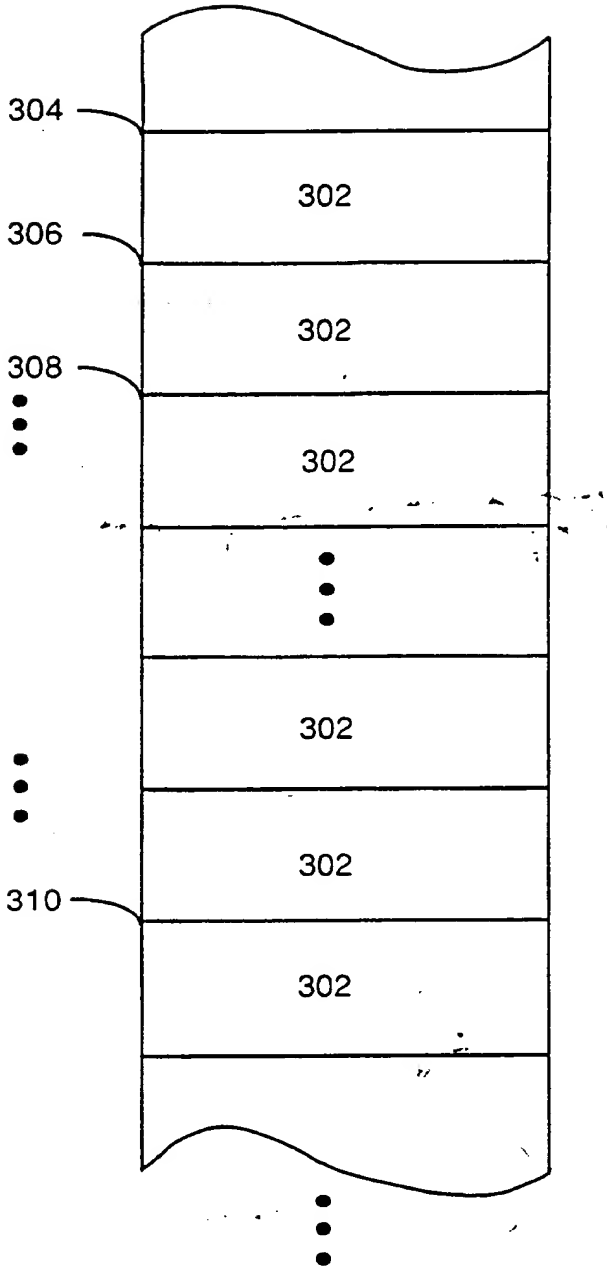


FIG. 3

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 93/10836

A. CLASSIFICATION OF SUBJECT MATTER
IPC 5 G06F9/26 G06F12/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 5 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
E	US,A,5 274 829 (HOTTA ET AL.) 28 December 1993 see the whole document	1-8, 10-12
X	& PATENT ABSTRACTS OF JAPAN vol. 012, no. 366 (P-765)30 September 1988 & JP,A,63 116 236 (HITACHI, LTD.) 20 May 1988 see abstract	1-8, 10-12
X	PATENT ABSTRACTS OF JAPAN vol. 012, no. 295 (P-743)11 August 1988 & JP,A,63 068 930 (HITACHI LTD.) 28 March 1988	1-3,5,7, 8,10
Y	see abstract	4,11,12
X	EP,A,0 075 741 (SIEMENS AKTIENGESELLSCHAFT) 6 April 1983 see the whole document	1-5,7,8

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 February 1994	15.03.94

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Daskalakis, T

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 93/10836

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	DE,A,24 42 014 (COMPAGNIE HONEYWELL BULL) 13 March 1975 see page 14, paragraph 3 - page 15, paragraph 1 ---	4,11,12
A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 32, no. 3B , August 1989 , NEW YORK US pages 314 - 316 'Three-Level Memory Structure for Performance-Oriented Machine Code Placement' see page 1 - page 2, paragraph 5 ---	1-4,6-8
A	PATENT ABSTRACTS OF JAPAN vol. 011, no. 178 (P-584)9 June 1987 & JP,A,62 008 243 (MATSUSHITA ELECTRIC IND. CO. LTD.) 16 January 1987 see abstract -----	1-3

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 93/10836

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US-A-5274829	28-12-93	JP-A- 63116236	20-05-88
EP-A-0075741	06-04-83	DE-A- 3138971	21-04-83
		JP-A- 58070357	26-04-83
DE-A-2442014	13-03-75	FR-A- 2336058	15-07-77
		GB-A- 1478489	29-06-77
		JP-C- 1279853	13-09-85
		JP-A- 50056136	16-05-75
		JP-B- 60005976	15-02-85